In this report, the basic steps of how I implemented MapReduce to obtain desired results are explained.

Mapper:

```
public static class MyMapper extends Mapper<Object, Text,Text,Text>{
    private Text word = new Text();
    private Text occur = new Text();

    @Override
    public void map(Object key,Text value, Context context) throws IOException, InterruptedException{
        FileSplit fileSplit = (FileSplit)context.getInputSplit();
        String filename = fileSplit.getPath().getName();
        String line = value.toString();
        String[] splited = line.split(" ");
        int n = Integer.parseInt(context.getConfiguration().get("ngram"));

        for (int i = 0; i <= splited.length - n + 1; i ++) {
            List<String> temp = new ArrayList<String>();
            if((n + i-1) < splited.length){
                for (int j = i;j < (n + i);j += 1){
                    temp.add(splited[j]);
                }
                word.set(String.join(" ", temp));
                occur.set(filename);
                context.write(word,occur);
            }
        }
    }
```

The first highlight place is the FileSplit class which I used in the Map function to obtain corresponding file name. The second circled part is the main part of the Map function where I used to two loops to find all ngrams. For each ngram found, I used it as a key and the file name as value in order to form the (key, value) pair. Thus, the output key and value class of Mapper are set to be Text.

Reducer:

```
public static class MyReducer extends Reducer<Text, Text, Text, Text>{
    private Text result = new Text();
    @Override
    protected void reduce(Text key, Iterable<Text> values,Context context) throws IOException, InterruptedException
        int count = 0;
        int occur = Integer.parseInt(context.getConfiguration().get("occur"));
        List<String> temp = new ArrayList<String>();
        String cc = "";
        String files = "";
        while (values.iterator().hasNext()) {
            cc = values.iterator().next().toString();
            if(!temp.contains(cc)){
                temp.add(cc);
            }
            count++;
        }
        Collections.sort(temp);
        files = String.join(" ", temp);

        if(count >= occur){
            result.set(Integer.toString(count) + "  " + files);
            context.write(key, result);
        }
    }
}
```

The circled part is the main part of the reduce function. Taking the outputs from mapper in the form of (key: ngram, value: Text object which contains all files that contain particular ngram). Then, the reduction process is to count the number of files and concatenate that number with all the file names from the values to a new Text object, and set that Text object as the final output value with corresponding ngram as the key.

Main:

In the main function, in order to obtain the arguments in mapper and reducer function:

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();
    conf.set("ngram", args[0]);
    conf.set("occur", args[1]);
    Job job = Job.getInstance(conf, Assignment1.class.getSimpleName());
    job.setJarByClass(Assignment1.class);
    FileInputFormat.addInputPaths(job, args[2]);
    job.setMapperClass(MyMapper.class);
    job.setMapOutputKeyClass(Text.class);
    job.setMapOutputValueClass(Text.class);
```